PROJECT-DEEPRL ARM MANIPULATION

Abstract – In this paper, the topic is a solution of the Deep RL Arm Manipulation project as part of the Robotics Nanodegree. The goal of the project is to set up and optimize a reinforcement learning framework. A Deep Q-Network learns to control a robotic arm in a simulated Gazebo environment. The reinforcement learning agent is given the task of touch a target object.

Objective 1: Have any part of the robot arm touch the object of interest, with at least a 90% accuracy.

Objective 2: Have only the gripper base of the robot arm touch the object, with at least a 80% accuracy.

The project tasks were to implement parts of the ArmPlugin.cpp file. The reward function was to be designed, and the DQN hyper-parameters were to be tuned.

---

1. Introdution

Before running simulation experiments, the following parts of the **ArmPlugin.cpp** file were implemented.

- The subscription to camera and collision topics.
- The creation of the DQN agent using learning hyperparameters defined as constants in the file.
- The definition of collision checks to determine which robot arm link touched the target object.
- Both velocity control and position control of the robot were implemented.

❖ Criteria for ground contact of the arm were defined. The *z* coordinates of the gripper link bounding box were used. The threshold ground level was set to accommodate for the inaccuracy of the distance calculation.

Several experimental runs of the simulation were performed, testing the reward function and tuning the DQN parameters.

1.1 Reward Function

A reward system was set up in order to teach the robotic arm how to reach for the target object. Positive and negative rewards are defined as numeric constants **REWARD WIN** and **REWARD LOSS**. A win is issued for touching the prop with any link (**OBJECTIVE 1**) or the gripper base link (**OBJECTIVE 2**). The switch just determines which part of the arm is tested against the collision with the object for a positive reward. For both objectives, the same reward function was used using the position based control of its arm joints.

The three rewards are for the gripper hitting the ground, an interim reward based on the distance to the object and a reward for a collision between either the arm and the object or a collision between the gripper base and the object. In general **REWARD_WIN** is set at 10 and **REWARD_LOSS** is set at -10. The reward for the gripper hitting the ground was **maxEpisodeLength * REWARD_LOSS**. Since the max episode length is 100, the loss for the gripper hitting the ground was set at a discouraging large -1000. The interim reward based on the distance to the object was the same for both objectives. It is defined as:

rewardHistory = avgGoalDelta * REWARD_WIN * exp(-distGoal)

With:

avgGoalDelta = (avgGoalDelta * ALPHA) + (lastGoalDistance – distGoal) * (1 – ALPHA)

avgGoalDelta is essentially a smoothing function. It weights the importance of the past avgGoalDelta by a factor of **ALPHA** and the new delta distance between the

distance to the goal now and the previous distance to the goal by a factor of 1 - **ALPHA**. This helps stabilize the movement of the arm. If **ALPHA** is too small, the arm is much more unstable (moving back and forth) in its pursuit of the objective.

The **rewardHistory** takes this smoothed delta distance and multiplies it by **REWARD_WIN** and an exponential function gives higher values to shorter distances to the goal encouraging the optimizer to move toward the goal quicker.

The reward for the collision between the arm and the object or the gripper base and the object (determined by the tested objective) is given by:

(maxEpisodeLength – episodeFrames) * REWARD_WIN

This reward encourages the optimizer to find a path in the shortest period of time as the less number of frames to accomplish the objective corresponds to a higher reward.

1.2 DQN Hyperparameter Tuning

The following is a list of hyperparameters:

1.2.1 **OPTIMIZER "Adam"**

The other one tested was RMSprop but Adam performed better.

1.2.2 **EPS_END 0.01f**

This helps determine how fast the learning trails off towards the end of the episode. A smaller number was chosen from trial and error because most of the learning that made the arm successful was done towards the beginning. A higher number at the end caused the arm to try and learn more when it was not needed which affected the accuracy negatively.

1.2.3 **EPS_DECAY 70**

This number was also made smaller similar to the reason done so for **EPS_END**. A smaller decay number allowed the optimizer to learn more towards the beginning of the episode before trailing off quicker towards the end. This was done because most of the learning successfully contributing to good arm performance was done toward the beginning.

1.2.4 **BATCH_SIZE 8**

This number was kept as presented. Changing this experimentally didn't appear to add any extra benefits.

1.2.5 **INPUT_WIDTH 64 AND INPUT_HEIGHT 64**

These numbers were made smaller because a smaller resolution was all that was needed and used less memory and computational power.

1.2.6 **LEARNING_RATE 0.03f**

This number was increased from 0.01. It seemed to perform better with more learning chance at the beginning of the episode.

1.2.7 **REPLAY_MEMORY 10000 for OBJECTIVE 1 and 50000 for OBJECTIVE 2**

The original 10000 worked fine experimentally for objective 1, but for objective 2 the cyclic replay buffer was increased by a factor of 5 because it was harder for the arm to hit a smaller target (the base) to the object and therefore when it did it needed to be stored in the cyclic buffer to be sampled from again for a longer period of time. Otherwise, it may not get a chance to be resampled as easily.

1.2.8 **USE_LSTM true and LSTM_SIZE 256**

This was used only for objective 1. When using LSTM, if a good path is found to the goal, LSTM allows the arm to repeatedly follow the same successful behavior. LSTM doesn't work as well if a good path is not found early on because LSTM will often repeat a less than optimal path again instead of exploring a new solution. Using LSTM did not work well experimentally for objective 2.

1.2.9 **ALPHA 0.8 for OBJECTIVE 1 and 0.7 for OBJECTIVE 2**

Having a slightly higher ALPHA for objective 1 than objective 2 seemed to work a little better. For objective 1 the path was found more consistently and for objective 2 it seemed to give the arm a little bit more room to move around and find an optimal solution.

2. Results

2.1 Objective 1

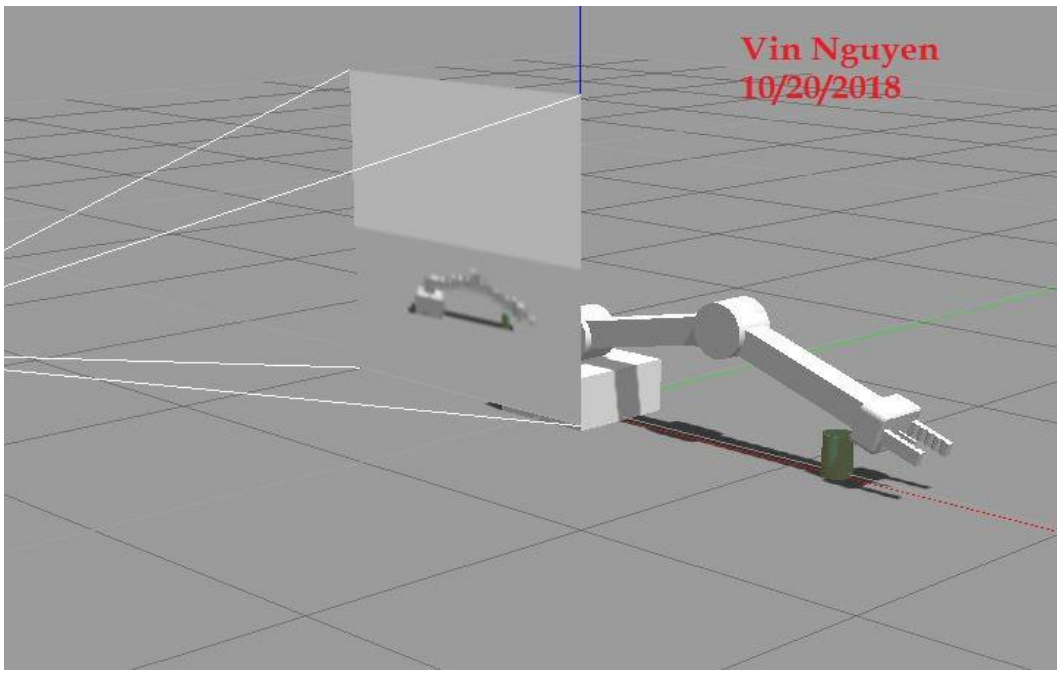

Figure 1. OBJECTIVE 1

A 90% win rate was achieved at the 160 iterations and continued to increase in accuracy. Using LSTM here was beneficial to keep the arm choosing the same optimal path. We can see below image:

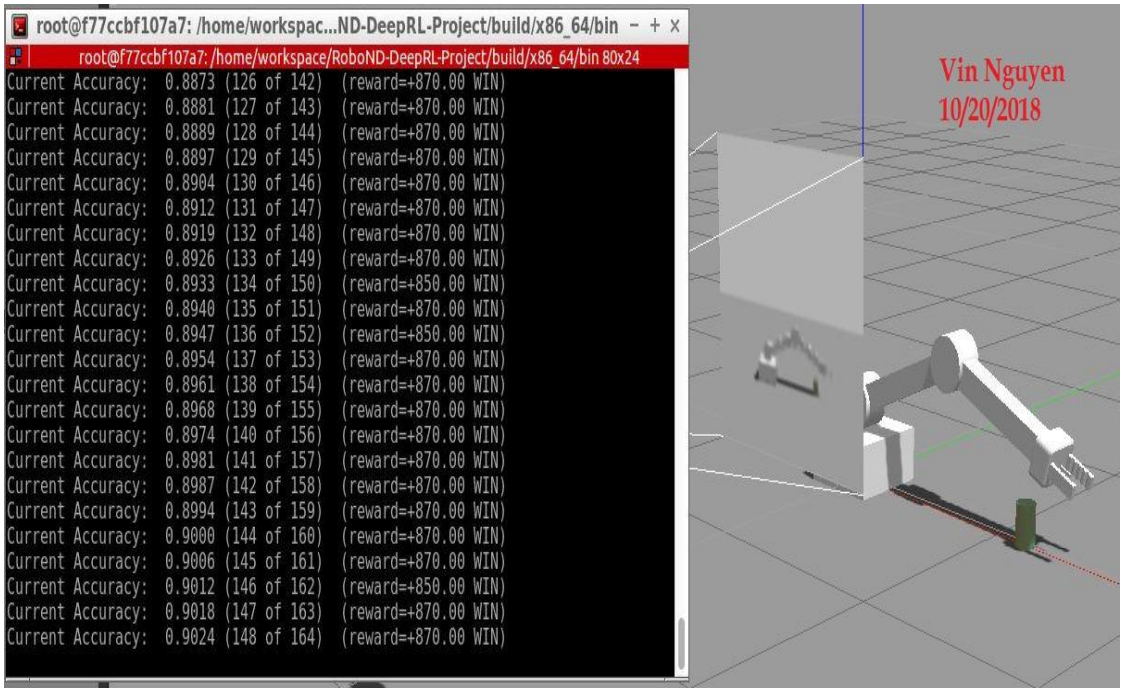

Figure 2. Accuracy of objective 1

2.2 Objective 2

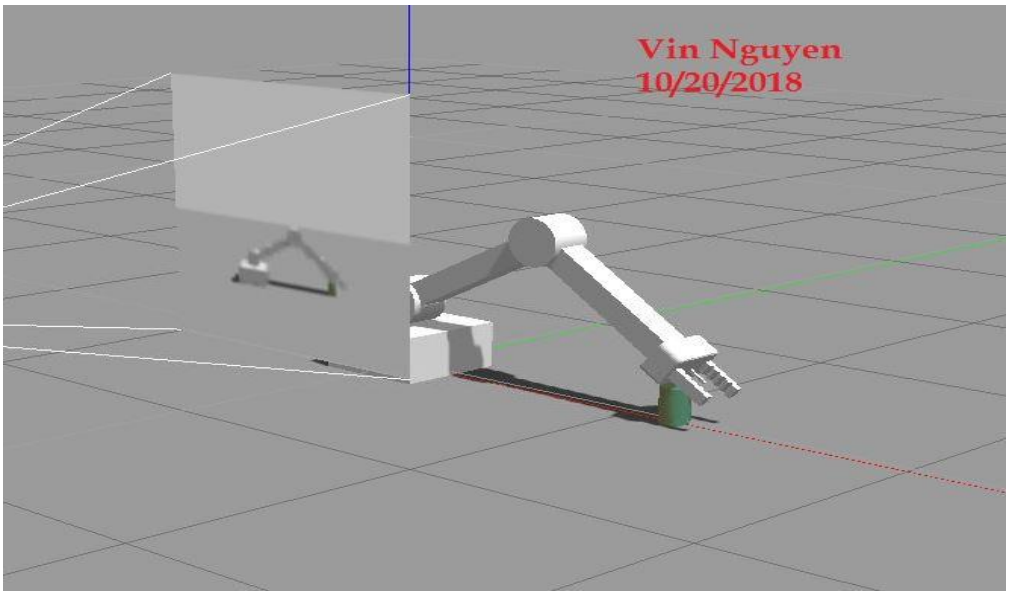

Figure 3. OBJECTIVE 2

An 80% win rate was achieved on the 260 iteration and continued to increase in accuracy. Increasing the **REPLAY_MEMORY** was very beneficial for this objective. We can see below image:

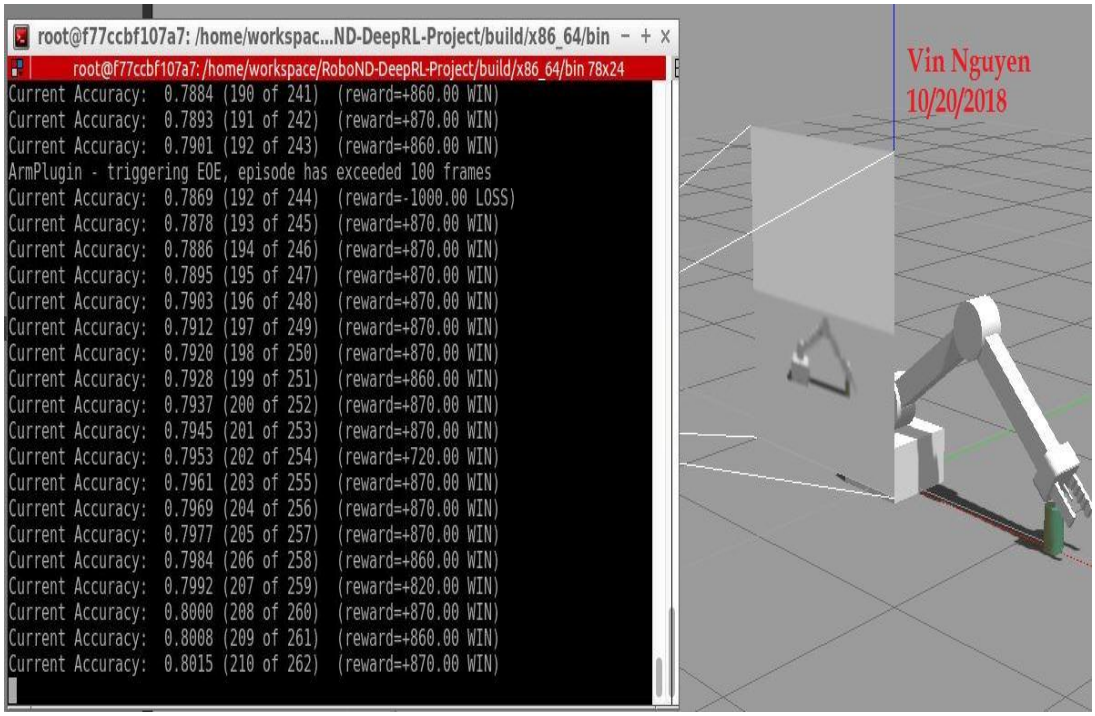

Figure 4. Accuracy of objective 2

The only real difficulty noted in the performance of the DQN agent was when one of its first random paths chosen resulted in the top joint having an angle closer to 90 than 180. The incremental reward function was not as effective in these cases. Often it would get stuck very close to the object on the near side but not actually touch the object. The training took much longer in these cases to overcome initial unsuccessful attempts. This was especially true with objective 1, but was true to some degree for object 2 as well.

3. Future Work

The reinforcement-learning framework can be improved further by:

- ❖ Investigating that can create other reward functions able to achieve the objectives.
- ❖ May be to reward the robot when its top joint is straighter (closer to 180 than 90 degrees) to encourage it to more efficiently find an optimal path.
- ❖ Tuning again DQN hyperparameters like size of replay memory, and epsilon decay rate.

Also given more time, it would be helpful to read research papers to see how others came up with more effective interim reward functions for this type of problem.